UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/851,592	05/09/2001	Bhashyam Ramesh	9491	2588

26890 7590 09/15/2004

JAMES M. STOVER
NCR CORPORATION
1700 SOUTH PATTERSON BLVD, WHQ4
DAYTON, OH 45479

EXAMINER	
CAO, DIEM K	

ART UNIT	PAPER NUMBER
2126	

DATE MAILED: 09/15/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/851,592	RAMESH ET AL.	
	Examiner	Art Unit	
	Diem K Cao	2126	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *09 May 2001*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-38* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-38* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date *8/27/2001*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-38 are presented for examination.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1-4, 6-8, 10-19, 21-31 and 33-36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kleinman (U.S. 6,128,640) in view of Connelly et al. (U.S. 5,706,515).

4. **As to claim 1**, Kleinman teaches a Unix operating system (UNIX, Posix; col. 1, lines 17-22 and col. 3, lines 50-54), a plurality of execution entities (threads; col. 3, lines 65-67), an event control module adapted to create an event (The alternative API would consist of constructing a Thread_Exit_Event object; col. 7, lines 7-10), one or more of the execution entities adapted to wait on the event (The application can add ... to be waited for in a container Alert; col. 7, lines 39-41 and Each of M threads ... different events; col. 4, lines 15-20), and a controller (processor) adapted execution entities to awaken the one or more execution entities by signaling the execution entities if the event occurs (When a thread blocks on a single Alert object ... these events occur; col. 4, lines 31-35, 50-53 and Because the thread is waiting ... unblocking the thread; col. 8, lines 45-48).

5. However, Kleinman does not teach an event having a state, and awaken the one or more execution entities by signaling the execution entities when the event state changes to a predetermined state. Connelly teaches an event having a state (the purpose of an event object is to store a state value that is equal to either "signaled" or "unsignalled"; col. 3, lines 40-43), and awake the one or more execution entities by signaling the execution entities when the event state changes to a predetermined state (Notification can be used ... is available for use; col. 2, lines 60-62 and col. 3, lines 20-55).

6. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and Connelly because it would provides a method to handle the event object when the event occurs.

7. **As to claim 2**, Kleinman teaches the event control module is adapted to define an event object representing the event (All event objects representing any type of event; col. 4, lines 46-47), the event object associated with a collection having one or more entries corresponding to one or more execution entities waiting on the event represented by the event object (The application can add the event ... in a container Alert; col. 7, lines 39-41 and The Thread_Exit_Event ... it represents exits; col. 11, Appendix C and (thread->alert).notify_all(); col. 12, Appendix C).

8. However, Kleinman does not teach the event object associated with a queue. Kleinman teaches a Container Alert Object contains a collection of Alert objects, each of which

corresponds to an event (col. 4, lines 28-30), the AlertCollection class encapsulates the details of a collection of Alerts (col. 5, lines 20-25), and the AlertCollection class supports array-like retrieval. It is well know in the art that the queue could be implemented as an array.

9. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and well known technique to implement a queue to store the Alert objects that associated with an event because queue provides a method to store data that can be retrieved or access at any time by many ways.

10. **As to claim 3**, Kleinman teaches the event control module is adapted to further create one or more second objects (An application can create a container Alert object ... via the own routine; col. 6, lines 25-28 and Alert (), Alert (const Alert &); col. 9, class Alert), wherein each entry comprises a link to a corresponding second object (an Alert object 510 ... in the container AlertCollection 520 of the Alert object; col. 5, lines 40-50), each execution entity to sleep on an associated second object to wait on the event (a thread blocks on a single Alert object; col. 4, lines 31-32 and the thread is waiting on the container Alert collection; col. 8, lines 45-48).

11. **As to claim 4**, Kleinman teaches each second object is defined by a condition variable (pthread_cond_t cond; col. 9, class Alert, line 15).

12. **As to claim 6**, Kleinman teaches each second object is defined by a condition variable and a mutex (pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16).

13. **As to claim 7**, Kleinman does not teach each event object contains an indication of the state of the event. Connelly teaches each event object contains an indication of the state of the event (the purpose of an event object is to store a state value that is equal to either "signaled" or "unsignalled"; col. 3, lines 40-43).

14. **As to claim 8**, Kleinman does not teach the indication has a first state to indicate that the event has been signaled and a second state to indicate that the event has not been signaled, the predetermined state comprising the first state. Connelly teaches the indication has a first state to indicate that the event has been signaled and a second state to indicate that the event has not been signaled (the purpose of an event object is to store a state value that is equal to either "signaled" or "unsignalled"; col. 3, lines 40-43), the predetermined state comprising the first state (Notification can be used ... is available for use; col. 2, lines 60-62 and col. 3, lines 20-55).

15. **As to claim 10**, Kleinman teaches the event control module is adapted to further define at least another event, one of the execution entities to wait on the plural events (the process can set a timer, and add the event corresponding to a timer's expiration to the same heterogeneous list of events to be waited for; col. 7, lines 55-67 and col. 8, lines 45-48).

16. **As to claim 11**, Kleinman teaches each of the event is represented by a corresponding event object (All event objects representing any type of event; col. 4, lines 46-47). However, Kleinman does not teach each event object having a first state to indicate that the event has been

signaled and a second state to indicate that the event has not been signaled. Connelly teaches each event object having a first state to indicate that the event has been signaled and a second state to indicate that the event has not been signaled (the purpose of an event object is to store a state value that is equal to either "signaled" or "unsignalled"; col. 3, lines 40-43).

17. **As to claim 12**, Kleinman does not teach comprising queues associated with corresponding event objects, each queue containing an entry corresponding to the one execution entity. Kleinman teaches a Container Alert Object contains a collection of Alert objects, each of which corresponds to an event (col. 4, lines 28-30), the AlertCollection class encapsulates the details of a collection of Alerts (col. 5, lines 20-25), and the AlertCollection class supports array-like retrieval. It is well know in the art that the queue could be implemented as an array.

18. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and well known technique to implement a queue to store the Alert objects that associated with an event because queue provides a method to store data that can be retrieved or access at any time by many ways.

19. **As to claim 13**, Kleinman teaches the event control module is adapted to define a barrier object (An application can create a container Alert object ... via the own routine; col. 6, lines 25-28 and Alert (), Alert (const Alert &); col. 9, class Alert), the one execution entity to sleep on the barrier object to wait on the plural events (the thread is waiting on the container Alert collection rather than any single contained Alert; col. 8, lines 45-48 and a container Alert object ... to an

event; col. 4, lines 28-30), the entries of the queues each containing a link to the barrier object (The internal AlertCollection ... index offset; col. 6, lines 40-43).

20. **As to claim 14**, Kleinman teaches the barrier object is defined at least by a condition variable (pthread_cond_t cond; col. 9, class Alert, line 15).

21. **As to claim 15**, Kleinman teaches the barrier object is defined at least by a condition variable and a mutex (pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16).

22. **As to claim 16**, Kleinman teaches the event control module comprises a library (the alternative API; col. 7, lines 7-9 and Appendix A-C; col. 9-12).

23. **As to claim 17**, Kleinman teaches the execution entities comprise threads (threads; col. 3, lines 65-67).

24. **As to claim 18**, Kleinman teaches plural processes (process, subprocess; col. 3, line 65 – col. 4, line 2), each process associated with one or more threads (the threads that compose the process; col. 3, line 66), the event control module to create a local event to synchronize threads within a process and to create a global event to synchronize threads of different processes (exit of a subprocess, the exit of another thread, the completion of asynchronous file I/O ... Posix conditional; col. 4, lines 1-6).

25. **As to claim 19**, Kleinman teaches the global event comprises named event (Timer_Expiration_Event; col. 7, lines 60-67).

26. **As to claim 21**, Kleinman teaches generate event objects representing events used for synchronizing execution entities in the system (All event objects representing any type of event; col. 4, lines 46-47 and abstract), provide one or more entries associated with the event object (a thread creates an event Alert client for each event; col. 6, lines 57-62), each entry associated with a corresponding execution entity (for which the thread may desire to wait; col. 6, lines 57-62), and in response to the occurrence of an event of the event object, use the one or more entries to signal one or more corresponding execution entities (When a thread blocks on a single Alert object ... these events occur; col. 4, lines 31-35, 50-53 and Because the thread is waiting ... unblocking the thread; col. 8, lines 45-48).

27. However, Kleinman does not teach each event object having a state to indicate if the corresponding event has been signaled, and in response to the state of one of the event objects indicating the corresponding event has been signaled, use the one or more entries to signal one or more corresponding execution entities. Connelly teaches event object having a state to indicate if the corresponding event has been signaled (the purpose of an event object is to store a state value that is equal to either "signaled" or "unsignalled"; col. 3, lines 40-43), and in response to the state of one of the event objects indicating the corresponding event has been signaled, use the

one or more entries to signal one or more corresponding execution entities (Notification can be used ... is available for use; col. 2, lines 60-62 and col. 3, lines 20-55).

28. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and Connelly because it would provides a method to handle the event object when the event occurs.

29. **As to claim 22**, Kleinman teaches the instruction when executed cause the system to further create barrier objects (An application can create a container Alert object ... via the own routine; col. 6, lines 25-28 and Alert (), Alert (const Alert &); col. 9, class Alert), each execution entity waiting on a corresponding barrier object to wait on an event (the thread is waiting on the container Alert collection rather than any single contained Alert; col. 8, lines 45-48 and a container Alert object ... to an event; col. 4, lines 28-30).

30. **As to claim 23**, Kleinman teaches the instructions when executed cause the system to create barrier objects by defining each barrier object based on a condition variable according to a Unix operating system (An application can create a container Alert object ... via the own routine; col. 6, lines 25-28 and Alert (), Alert (const Alert &); col. 9, class Alert and pthread_cond_t cond; col. 9, class Alert, lines 15-16).

31. **As to claim 24**, Kleinman teaches the instructions when executed cause the system to create barrier objects by defining each barrier object based on a condition variable and mutex

according to a Unix operating system (An application can create a container Alert object ... via the own routine; col. 6, lines 25-28 and Alert (), Alert (const Alert &); col. 9, class Alert and pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16).

32. **As to claim 25**, Kleinman does not teach define a queue associated with each event object, the queue containing the one or more entries, the one or more entries pointing to one or more barrier objects.

33. Kleinman teaches a Container Alert Object contains a collection of Alert objects, each of which corresponds to an event (col. 4, lines 28-30), the AlertCollection class encapsulates the details of a collection of Alerts (col. 5, lines 20-25), and the AlertCollection class supports array-like retrieval. It is well know in the art that the queue could be implemented as an array (the internal AlertCollection ... index offset; col. 6, lines 40-43)

34. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and well known technique to implement a queue to store the Alert objects that associated with an event because queue provides a method to store data that can be retrieved or access at any time by many ways.

35. **As to claim 26**, Kleinman as modified teaches provide a routine associated with each event object, the routine to traverse each queue and to signal one or more barrier objects pointed

to by one or more entries in the queue (void *Thread_Exit_Event::wrapper (void *vp); col. 11, line 63 – col. 12, lines 13).

36. **As to claim 27**, Kleinman teaches provide plural events containing respective entries, each of the entries corresponding to one execution entity to enable the one execution entity to wait on plural events (Each of M threads … may represent N different events; col. 4, lines 15-20 and The application can add … to be waited for in a container Alert; col. 7, lines 39-41).

37. **As to claim 28**, Kleinman teaches providing one or more synchronization primitives (pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16), defining a first object based on the one or more synchronization primitives (Alert object, Alert class, pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16), defining an event object representing an event (All event objects representing any type of event; col. 4, lines 46-47 and Thread_Exit_Event object; col. 7, lines 8-10), and one of the execution entities sleeping one the first object to wait on the event (the thread is waiting on the container Alert … unblocking the thread; col. 8, lines 45-48).

38. However, Kleinman does not teach the event object having a state to indicate the event being signaled. Connelly teaches event object having a state to indicate if the corresponding event has been signaled (the purpose of an event object is to store a state value that is equal to either "signaled" or "unsignalled"; col. 3, lines 40-43).

39. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and Connelly because it would provides a method to handle the event object when the event occurs.

40. **As to claim 29**, Kleinman as modified teaches signaling the first object in response to the event object state indicating the event being signaled (All event object ... of which it is a member; col. 4, lines 46-52).

41. **As to claim 30**, Kleinman teaches signaling the first object comprises a routine associated with the event object signaling the first object (void *Thread_Exit_Event::wrapper (void *vp); col. 11, line 63 – col. 12, lines 13).

42. **As to claim 31**, Kleinman teaches providing the one or more synchronization primitives comprises providing one or more synchronization primitives defined in a Unix operating system (pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16 and UNIX, Posix; col. 1, lines 17-22 and col. 3, lines 50-54).

43. **As to claim 33**, Kleinman teaches providing the one or more synchronization primitives defined in a Unix operating system (pthread_mutex_t mutex, pthread_cond_t cond; col. 9, class Alert, lines 15-16 and UNIX, Posix; col. 1, lines 17-22 and col. 3, lines 50-54).

44. **As to claim 34**, Kleinman teaches defining at least another event object representing another event, the one execution entity to add entries to the event objects to enable the one execution entity to wait on plural events (the process can set a timer, and add the event corresponding to a timer's expiration to the same heterogeneous list of events to be waited for; col. 7, lines 55-67 and col. 8, lines 45-48).

45. **As to claim 35**, Kleinman does not teach the one execution entity adding the entries to the event objects comprises adding entries to queues associated with the event objects. Kleinman teaches a Container Alert Object contains a collection of Alert objects, each of which corresponds to an event (col. 4, lines 28-30), the Alert objects has the functions make_member(), own () (col. 5, lines 65-66), the AlertCollection class encapsulates the details of a collection of Alerts (col. 5, lines 20-25), and the AlertCollection class supports array-like retrieval (col. 6, lines 40-43). It is well know in the art that the queue could be implemented as an array.

46. It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and well known technique to implement a queue to store the Alert objects that associated with an event because queue provides a method to store data that can be retrieved or access at any time by many ways.

47. **As to claim 36**, Kleinman does not teach adding the entries to the queues comprises adding a pointer to the first object. Kleinman teaches the C++ programming language is used to implement the system, and it is well known that object is passed by reference (pointer) is

supported in the C++. It would have been obvious adding the entries to the queues comprises adding a pointer to the first object.

48. Claims 5 and 32 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kleinman (U.S. 6,128,640) in view of Connelly et al. (U.S. 5,706,515) further in view of Cutler et al. (U.S. 5,598,562).

49. **As to claim 5**, Kleinman does not teach the controller signals each thread by signaling the condition variable. Kleinman teaches the controller signals each thread by signaling the second object (All event objects ... by one of these containers; col. 4, lines 46-56). Cutler teaches the controller signals each thread by signaling the condition variable (Kernel synchronization primitives ... A timer is changed to signaled; col. 27, lines 3-30). It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and Cutler because it provides a method for synchronization to help coordinate access to resources or data without having to modify the wait service routines and the scheduler in the operating system's kernel.

50. **As to claim 32**, see rejections of claims 4-5 above.

51. Claims 9 and 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kleinman (U.S. 6,128,640) in view of Connelly et al. (U.S. 5,706,515) further in view of Kakivaya et al. (U.S. 6,546,443 B1).

52. **As to claim 9**, Kleinman does not teach each event object has a type indication to indicate whether the event object state is to be automatically reset to the second state from the first state once the event has been signaled or to be manually reset to the second state from the first state by an explicit action. Kakivaya teaches each event object has a type indication to indicate whether the event object state is to be automatically reset to the second state from the first state once the event has been signaled or to be manually reset to the second state from the first state by an explicit action (The illustrated event objects support tow basic event type ...is manually reset). It would have been obvious to one of ordinary skill in the art at the time the invention was made to combine the teaching of Kleinman and Kakivaya because it provides a method to send the awake one thread or more that are waiting on the event.

53. **As to claim 20**, Kleinman does not teach a plurality of nodes, each node comprising one or more of the plurality of execution entities. Kakivaya teaches a plurality of nodes, each node comprising one or more of the plurality of execution entities (The invention may be practiced ...distributed computing environments; col.4, line 64 – col. 5, line 11 and threads 320; col. 9, lines 15-18 and Fig. 3).

54. Claims 37-38 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kleinman (U.S. 6,128,640).

55. **As to claim 37**, Kleinman teaches a Unix operating system (UNIX, Posix; col. 1, lines 17-22 and col. 3, lines 50-54), a plurality of execution entities (threads; col. 3, lines 65-67), a storage module containing an event class (a data storage medium 130; col. 3, lines 44-46 and sample event class, Thread_Exit_Event; col. 11-12, Appendix C), and a processor (processor; col. 3, lines 55-64) adapted to execute the event class to provide an event-based synchronization mechanism comprising one or more events on which the plural execution entities are able to sleep (col. 4, lines 15-53 and col. 7, lines 8-10, 38-41 and col. 8, lines 45-48).

56. However, Kleinman does not teach an event library. Because there are multiple event classes in the system, it would have been obvious to one of ordinary skill in the art to put all the event classes in the library for easier maintenance and use.

57. **As to claim 38**, Kleinman teaches plural processes (process, subprocess; col. 3, line 65 – col. 4, line 2), each process associated with one or more threads (the threads that compose the process; col. 3, line 66), the event control module to create a local event to synchronize threads within a process and to create a global event to synchronize threads of different processes (exit of a subprocess, the exit of another thread, the completion of asynchronous file I/O ... Posix conditional; col. 4, lines 1-6).

Conclusion

58. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

- Woft (U.S. 5,961,584) teaches "System for managing internal execution threads".

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Diem K Cao whose telephone number is (703) 305-5220 or (571) 272-3760 (after November 1st 2004). The examiner can normally be reached on Monday - Thursday, 9:00AM - 5:00PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai An can be reached on (703) 305-9678 or (571) 272-3756 (after November 1st 2004). The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Any response to this action should be mailed to:
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Diem Cao

MENG-AL T. AN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100